AFRL-VA-WP-TP-2003-312

TRANSPORT LAYER ABSTRACTION IN EVENT CHANNELS FOR EMBEDDED SYSTEMS

David Sharp
Edward Pla
Ravi Pratap M
Ron K. Cytron

MAY 2003

Approved for public release; distribution is unlimited.

AIR VEHICLES DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7542

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
May 2003	Conference Paper Preprint	

4. TITLE AND SUBTITLE
TRANSPORT LAYER ABSTRACTION IN EVENT CHANNELS FOR EMBEDDED SYSTEMS

5a. CONTRACT NUMBER
F33615-00-C-3048
(See block 13)

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
69199F

6. AUTHOR(S)
David Sharp and Edward Pla (The Boeing Company)
Ravi Pratap M and Ron K. Cytron (Washington University)

5d. PROJECT NUMBER
ARPF

5e. TASK NUMBER
04

5f. WORK UNIT NUMBER
23

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The Boeing Company
P.O. Box 516
St. Louis, MO 63166

Washington University
Department of Computer Science and Engineering
St. Louis, MO 63130

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Vehicles Directorate
Air Force Research Laboratory
Air Force Materiel Command
Wright-Patterson Air Force Base, OH 45433-7542

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/VACC

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-VA-WP-TP-2003-312

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

To be presented at the Languages, Compilers, and Tools for Embedded Systems Conference, San Diego, CA June 11-13, 2003.
The other two contract numbers on this report are: F33615-00-C-1697 and F33615-97-D-1155.
Although originally created in color, this report was submitted to DTIC in black and white.

14. ABSTRACT *(Maximum 200 Words)*
As embedded systems increase in complexity and begin to participate in distributed systems, the need for middleware in the building such systems becomes imperative. However, the use of middleware that fully implements such standards can impose a significant increase in footprint for an application, making it unsuitable for use in embedded systems. We consider the use of a standard CORBA event channel in a setting where distribution and inter-language support are unnecessary. We report our experience in applying aspects to abstract the transport layer (CORBA) of the event channel into a selectable feature. Thus, enabling or disabling CORBA for a specific application can be decided at build-time, by merely selecting CORBA as a feature. We describe the patterns used to achieve this abstraction and present footprint and throughput results showing the effect CORBA on automatically derived subsets of the event channel.

15. SUBJECT TERMS
Algorithms, Design, Experimentation, Performance, AOP, CORBA, middleware, embedded systems, subsetting, event service, software composition, transport abstraction

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Daniel Schreiter
Unclassified	Unclassified	Unclassified	SAR	16	19b. TELEPHONE NUMBER *(Include Area Code)* (937) 255-8291

Transport Layer Abstraction in Event Channels for Embedded Systems *

Ravi Pratap M
Department of Computer Science and Engineering
Washington University
St. Louis, MO 63130
ravip@cse.wustl.edu

Ron K. Cytron
Department of Computer Science and Engineering
Washington University
St. Louis, MO 63130
cytron@acm.org

David Sharp
The Boeing Company
P.O. Box 516
St. Louis, MO 63166
david.sharp@boeing.com

Edward Pla
The Boeing Company
P.O. Box 516
St. Louis, MO 63166
edward.pla@boeing.com

ABSTRACT

As embedded systems increase in complexity and begin to participate in distributed systems, the need for middleware in building such systems becomes imperative. However, the use of middleware that fully implements such standards can impose a significant increase in footprint for an application, making it unsuitable for use in embedded systems. We consider the use of a standard CORBA event channel in a setting where distribution and inter-language support are unnecessary. We report our experience in applying *aspects* to abstract the transport layer (CORBA) of the event channel into a selectable feature. Thus, enabling or disabling CORBA for a specific application can be decided at build-time, by merely selecting CORBA as a feature. We describe the patterns used to achieve this abstraction and present footprint and throughput results showing the effect of CORBA on automatically derived subsets of the event channel.

Categories and Subject Descriptors

C.2.4 **[Computer-Communication Networks]**: Distributed Systems; C.3 **[Computer Systems Organization]**: Special-Purpose and Application-Based Systems—*Real-time and embedded systems*; D.1.m **[Programming Techniques]**: Miscellaneous

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

AOP, CORBA, middleware, embedded systems, subsetting, event service, software composition, transport abstraction

*This work was sponsored by the DARPA Information Exploitation Office Program Composition for Embedded Software program under contracts F33615-00-C-1697 and F33615-00-C-3048, administered by the Air Force Research Laboratory (AFRL) Real-Time Java for Embedded Systems program, Wright-Patterson Air Force Base (WPAFB), Information Directorate, under contract F33615-97-D-1155.

1. INTRODUCTION

Distributed systems have relied on common interfaces to achieve a high-degree of inter-operability, reliability and reusability. The emergence of standards such as the Common Object Request Broker Architecture (CORBA) [20] and DCOM [18] and their wide spread use in building applications continues to underscore the importance of relying on such interfaces. As embedded systems are deployed in new scenarios such as distributed systems, the need to interact with and make use of existing standards is imperative. By using established communication standards between systems, it is possible to build effective and reusable embedded system components.

An Event Channel is a well-established, standard interface for decoupling the supplier and consumer of events in a distributed system [22] [29]. Event Channels can be made customizable using compositional approaches, such as feature specification using Aspect-Oriented Programming (AOP) [15] [16]. The footprint in such cases can be half of that required for a full-featured event channel when measurements exclude the size of the supporting Object Request Broker (ORB). However, what is of importance to an embedded application is the combined footprint of the event channel as well as the ORB. The footprint of a high-quality Event Service implementation such as the ADAPTIVE Communication Environment (ACE) ORB (TAO) Event Service in certain configurations can be quite high mostly due to the size of the ORB [14]. Clearly, The ACE ORB (TAO)'s footprint is the key concern for small-footprint event channels which need to be deployed in embedded systems with tight constraints and limited resources. While efforts are underway to create reduced-feature, small-footprint ORBs [9], there are compelling applications that do not need one or more of the following: distribution, inter-language support and real-time

guarantees.

This paper describes an application of aspect and component oriented programming techniques to an Event Service to concretely demonstrate and measure its benefits in one particular application. As such, however, it also provides evidence for the efficacy of applying these techniques to middleware for the more general class of distributed Real-Time Embedded Systems (RTES). We report on the experience of deploying an AOP-customized Event Channel, where the aspects also govern the use of CORBA and consequently, the inclusion or exclusion of an ORB. The resulting Event Channel has been used in a Real-Time Specification for Java^{TM} (RTSJ) setting for prototyping avionics middleware [26].

Our paper is organized as follows: Section 2 provides a brief overview of CORBA. Section 3 describes some of the motivation behind the need for developing flexible middleware. Section 4 describes current approaches to subsetting middleware such as event channels. Section 5 describes the architecture of the Framework for Aspect Composition for an EvenT channel (FACET) that we have built using AOP techniques and describes the challenges in abstracting the transport layer and the patterns used to solve them. Section 6 quantifies the benefits of our approach. Finally, Section 7 describes our planned future work applying aspects to flexible middleware components and services.

2. BACKGROUND

In this section, we give a brief overview of the CORBA reference model so as to provide a context for the work presented in subsequent sections.

2.1 Overview of the CORBA ORB Reference Model

CORBA Object Request Brokers (ORBs) allow clients to invoke operations on distributed objects without concern for object location, programming language, OS platform, communication protocols and interconnects, and hardware [13]. Figure 1 illustrates the key components in the CORBA reference model [21] that collaborate to provide this degree of portability, interoperability, and transparency. [1]

CORBA ORBs [20] allow clients to invoke operations on distributed objects without concern for the following issues:

- Object location: CORBA objects either can be collocated with the client or distributed on a remote server, without affecting their implementation or use.
- Programming language: The languages supported by CORBA include C, C++, Java [2], COBOL, and Smalltalk, among others.
- OS platform: CORBA runs on many OS platforms, including Win32, UNIX, MVS, and real-time embedded systems like VxWorks, Chorus, and LynxOS.
- Communication protocols and interconnects: The communication protocols and interconnects that CORBA run on include TCP/IP, IPX/SPX, FDDI, ATM, Ethernet, Fast Ethernet, embedded system backplanes, and shared memory.
- Hardware: CORBA shields applications from side effects stemming from differences in hardware, such as storage layout and data type sizes/ranges.

[1] This overview only focuses on the CORBA components relevant to this paper. For a complete synopsis of CORBA's components see [20].

[2] Java is a trademark of Sun Microsystems, Inc.

Figure 1 illustrates the components in the CORBA 2.x reference model, all of which collaborate to provide the portability, interoperability and transparency outlined above.

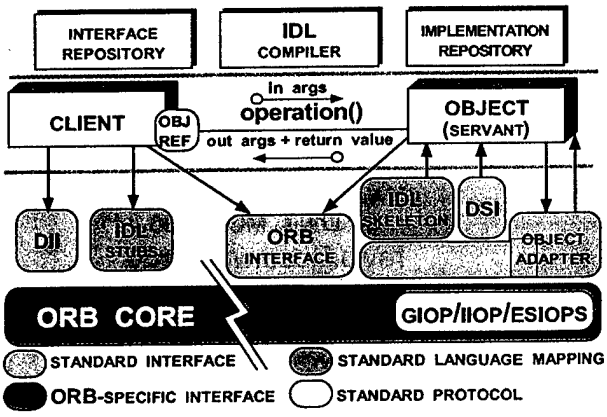

Figure 1: Components in the CORBA 2.x Reference Model

Each component in the CORBA reference model is outlined below:

- *Client:* A client is a *role* that obtains references to objects and invokes operations on them to perform application tasks. A client has no knowledge of the implementation of the object but does know its logical structure according to its interface. It also doesn't know of the object's location - objects can be remote or collocated relative to the client. Ideally, a client can access a remote object just like a local object, Figure 1 shows how the underlying ORB components described below transmit remote operation requests transparently from client to object.
- *Object:* In CORBA, an object is an instance of an Object Management Group (OMG) Interface Definition Language (IDL) interface. Each object is identified by an *object reference*, which associates one or more paths through which a client can access an object on a server. An *object ID* associates an object with its implementation, called a servant, and is unique within the scope of an Object Adapter. Over its lifetime, an object has one or more servants associated with it that implement its interface.
- *Servant:* This component implements the operations defined by an OMG IDL interface. In object-oriented (OO) languages, such as C++ and Java, servants are implemented using one or more class instances. In non-OO languages, such as C, servants are typically implemented using functions and `structs`. A client never interacts with servants directly, but always through objects identified by object references.
- *ORB Core:* When a client invokes an operation on an object, the ORB Core is responsible for delivering the request to the object and returning a response, if any, to the client. An ORB Core is implemented as a run-time library linked into client and server applications. For objects executing remotely, a CORBA-compliant ORB Core communicates via a version of the General Inter-ORB Protocol (GIOP), such as the Internet Inter-ORB Protocol (IIOP) that runs atop the TCP transport protocol.

- *ORB Interface:* An ORB is an abstraction that can be implemented various ways, *e.g.*, one or more processes or a set of libraries. To decouple applications from implementation details, the CORBA specification defines an interface to an ORB. This ORB interface provides standard operations to initialize and shut down the ORB, convert object references to strings and back, and so on.

- *IDL Stubs and Skeletons:* IDL stubs and skeletons serve as a "glue" between the client and servants, respectively, and the ORB. Stubs implement the *Proxy* pattern [8] and marshal application parameters into a common message-level representation. Conversely, skeletons implement the *Adapter* pattern [8] and demarshal the message-level representation back into typed parameters that are meaningful to an application.

- *IDL Compiler:* An IDL compiler transforms OMG IDL definitions into stubs and skeletons that are generated automatically in an application programming language, such as C++ or Java. In addition to providing programming language transparency, IDL compilers eliminate common sources of network programming errors and provide opportunities for automated compiler optimizations [6].

- *Object Adapter:* An Object Adapter is a composite component that associates servants with objects, creates object references, demultiplexes incoming requests to servants, and collaborates with the IDL skeleton to dispatch the appropriate operation upcall on a servant. Even though different types of Object Adapters may be used by an ORB, the only Object Adapter defined in the CORBA specification is the Portable Object Adapter (POA).

3. MOTIVATION

One of the special challenges associated with embedded systems is supporting their great diversity. Even within the very closely related set of avionics systems associated with the Boeing Bold Stroke product line software initiative, systems may have anywhere from one to ten processors, may run on Versa Module Europe (VME) and/or fiber channel based interconnects, may have one or more languages, and may run on a range of different operating systems. When these characteristics are taken together, the simplest deployed systems are single processor applications written completely in C++ without any interprocess communication, and the most complex ones are multiple VME backplanes connected by fiber channel, each with multiple processors, also written entirely in C++. There are also mixed C++ and Ada-based distributed systems. All of these are real-world systems, deployed in practice.

Even within a single product, embedded systems often impose significant resource constraints. Resource constraints may stem from limitations on size, weight, power, cost, aging hardware or other factors. Small consumer-devices such as cell phones and hearing aids emphasize size, weight, power, and cost factors. Larger, mass-produced systems such as automotive electronics are more focused on cost. More complex systems such as avionics are expected to be operational for a long life due to their expense.

Developing a single framework and implementation software that can be reused across a range of products as in Bold Stroke however, poses significant additional challenges. A component-based application architecture was created to provide a configurable and composable application capability [26]. ACE [24], TAO [4] and other architecture-specific services were created to provide a stable foundation for these applications [5]. This middleware foundation is also highly configurable, but does not provide the same level of component-based granularity provided in the application. Furthermore, while the extensive use of compiler directives and macros, for instance, does provide a single code base for reuse across a wide range of platforms, the necessary scattering and tangling of the code needed for all of the variants can significantly obscure the code associated with one particular use. The limitations of this approach in providing middleware implementations that scale to meet the required feature set for a particular system without incurring overhead associated with unused features has led to substantial interest in production programs in component-oriented and aspect-oriented approaches to middleware.

Colleagues at the University of British Columbia on the Program Composition for Embedded Software (PCES) program have also shown how AOP approaches can alleviate some of the configurability limitations in the component based application for cases where customizations are inherently scattered and tangled with the baseline code.

Real-time developers are typically reluctant to adopt new technologies — including C, C++, and CORBA — because those technologies often ignore the needs of real-time systems. The use of Java in real-time applications is thus relatively new. While the use of Java as the sole language in large-scale real-time applications is unlikely in the near-term, efforts such as ours are focused on laying the foundation for such applications in the future.

4. RELATED WORK

The need to subset (or extend) middleware selectively has existed for some time. Before the emergence of AOP techniques to separate concerns, subsetting techniques made use of object-oriented patterns. In previous work, AOP techniques have been used to demonstrate a high degree of feature control and consequently, footprint management [15] [16] [14]. However, such work has not concentrated on abstracting the transport layer as a feature so as to allow an even greater level of customizability for a specific application. In the following, we provide some background to our use of AOP in doing transport layer abstraction in FACET.

4.1 Object-Oriented Subsetting Techniques

Developing middleware to be flexible in diverse environments often involves the following process [16]:

1. Building flexibility and extensibility around known variation points at the beginning.

2. Refactoring functionality out of the core middleware to extensions and adding flexibility as new features are added and as the footprint becomes too large.

Unfortunately, the first relies on a designer's ability to preconceive feature extension points. As this is impractical with all but the smallest frameworks, functionality inevitably gets added that will need to be refactored to extensions later. Quite a few object-oriented design patterns have been identified that document successful strategies to subsetting middleware. These include patterns such as Strategy [8], Interceptors, Extension Interface, Service Configurator and others [25]. Although such patterns have been used extensively in middleware such as ACE [24] and TAO [4], these come with some limitations:

1. They require additional infrastructure within the framework to support their presence. For example, Strategy and Interceptors require method invocation hooks to be placed at key locations throughout the code. From a programmer standpoint, these hooks and the additional infrastructure lessen the readability and maintainability of the code.

2. If the locations where subsetting should have occurred are not preconceived, time consuming refactoring may be needed to extract functionality into separate libraries.

3. The hooks and infrastructure themselves can lead to degradations in performance and increase in footprint size if some or all of them are not used.

4.2 Separation of Concerns using AspectJ

AOP [17] is a software development paradigm that enables one to separate concerns that crosscut sets of classes and encapsulate those concerns in self-contained modules called *aspects*. The AspectJ [28] programming language adds AOP constructs to Java [2] and uses the following terminology. Within an aspect, the locations at which code should be applied are defined using *pointcuts*. Each pointcut is made up of one or more *joinpoints*, which are well-defined points in the execution of a program. The code applied at a pointcut is called *advice*. In addition to applying advice, languages supporting AOP often allow new methods or other language features to be *introduced* into existing classes. It is also possible to change the inheritance hierarchy of a class by changing the list of interfaces it implements or the classes it derives from.

In previous work [15] [16], this powerful, new programming paradigm has been applied to build software using the compositional approach by building a core of basic functionality and then codifying all additional features into separate aspects. Since the transport layer used in the event delivery mechanism is a crosscutting concern for the set of classes implementing the functionality of the event channel, it would be possible to abstract this into a separate aspect such that the inclusion or exclusion of the same produces an event channel with the desired functionality.

5. IMPLEMENTATION

In this section, we describe our AOP approach for compositional construction of an event channel. We focus only on the particulars of our implementation that are relevant to the transport layer and its abstraction. We omit details concerning other features, performance of those features, and our testing framework [15] [16] [14].

5.1 Architecture of FACET

FACET is an implementation of a CORBA [20] *event channel* that uses AOP to achieve a high level of customizability. Its functionality is based on features found in the OMG Event Service [22], the OMG Notification Service [19] [11], and the TAO Real-time Event Service [23] [12].

An event channel is a common middleware framework that decouples event suppliers and consumers. The event channel acts as a mediator through which all events are transported. Figure 2 shows the main participants in an event-channel framework. At its simplest,

- Suppliers push events to the event channel
- The event channel applies any filtering, correlation or other specified features to the events
- The event channel pushes appropriate events to consumers.

Event channel implementations differ in the types of events that they handle and in the processing and forwarding that occurs within the channel.

FACET is separated into a *base* and a *set of selectable features*. The base represents a fundamental and indivisible level of functionality. Each feature adds a structural and/or functional enhancement

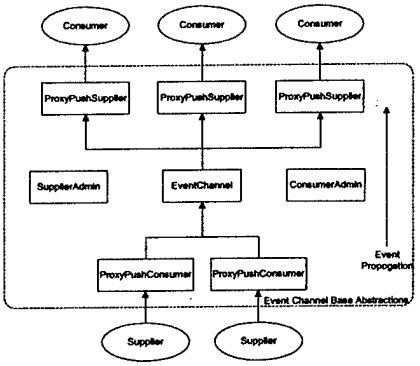

Figure 2: Main participants in an event channel.

to the base. Moreover, a feature can affect the structure and function of other features. AOP language constructs are then used to integrate or to weave feature code into the appropriate places in the base and selected features.

The construction of FACET follows a bottom-up approach in which features are implemented as needed. As new requirements are presented, they are decomposed into one or more features. In the case of FACET, the features of several existing event services were selected one-by-one for incorporation. By using AOP techniques, the code for each of these features can then be weaved into the base at the appropriate points.

5.1.0.1 Lack of Transport Abstraction.

FACET was originally designed to use CORBA so that events can be sent to and received from remote consumers and suppliers. The advantages of CORBA include the ability to distribute the consumers and suppliers as well as to fashion their implementation for any language that maps to CORBA (*e.g.*, Java and C++). The language independence is obtained by specifying interface definitions via CORBA's IDL.

However, in certain usage scenarios where distribution and multi-language support is unnecessary, the use of CORBA becomes unnecessary. As described in Section 3, there are important examples of this case. In this situation, the underlying transport mechanism can be a simple method call, doing away with the need to make use of an ORB. Indeed, one form of this optimization is routinely used in the Bold Stroke event service, in the form of the Subscription and Filtering configuration [12]

Following our compositional approach [15], we sought to provide a standard interface for the event channel while making the use of CORBA optional as well. In other words, merely by selecting an EnableCorba or DisableCorba feature (which are mutually exclusive since it would make no sense to enable both) at build-time, it should be possible to obtain an event channel with the desired configuration.

In what follows, we describe the challenges in abstracting the use of CORBA in the event channel and how these were addressed by the use of AOP.

5.2 CORBA vs No-CORBA

Since FACET was originally designed to use CORBA, its interfaces were specified in CORBA IDL and the implementation code was written in terms of CORBA Stub and Skeleton classes [20]. For instance, the implementation of one method of the `SupplierAdmin` interface [22] looked like this:

```
public class SupplierAdminImpl
   extends SupplierAdminPOA {

   // Field and other method definitions

   public ProxyPushConsumer obtain_push_consumer()
   {
      ProxyPushConsumerImpl ppcImpl =
         new ProxyPushConsumerImpl (eventChannel_);

      try {

         org.omg.CORBA.Object obj =
            poa_.servant_to_reference (ppcImpl);

         ProxyPushConsumer ppc =
            ProxyPushConsumerHelper.narrow(obj);

         return ppc;

      } catch (Exception se) {
         // Appropriate code
      }

      return null;
   }
}
```

It is clear from the above that the challenge lies in separating the concerns related to CORBA from the actual event channel implementation code for implementing the various features offered by an event service (present in various other classes). In the following, we present how we solved this problem using what we call the "Placeholder" pattern.

5.3 Use of the Placeholder pattern

Consider the `ProxyPushConsumer` interface [22] of the event channel, when configured to push structured event data. In the case CORBA is in use, the IDL describing this interface would be :

```
interface PushConsumer {
    void push (in Event data);
    void disconnect_push_consumer ();
};

interface ProxyPushConsumer : PushConsumer {
    void
    connect_push_supplier (in PushSupplier supplier);
};
```

The IDL compiler when given the above would generate the necessary Stub and Skeleton classes [20]. Now to implement the `ProxyPushConsumer` interface, the event channel's implementation would include a `ProxyPushConsumerImpl` class with the following definition:

```
public class ProxyPushConsumerImpl
       extends ProxyPushConsumerPOA {

       // Appropriate implementation

}
```

However, in the case CORBA is not needed, the interfaces can directly be specified in Java:

```
public interface PushConsumer {
    public void push (Event data);
    public void disconnect_push_consumer ();
}

public interface ProxyPushConsumer
       implements PushConsumer {

    public void
    connect_push_supplier (PushSupplier supplier);
}
```

And the corresponding implementation of the `ProxyPushConsumer` interface would be:

```
public class ProxyPushConsumerImpl
       implements ProxyPushConsumer {

       // Appropriate implementation

}
```

This idea recurs for all interfaces exposed by the event channel and is a concern which is independent of the manner of implementation. To address this issue, we make use of what we call the "Placeholder Class" pattern. This pattern makes use of a single, empty base class which the implementation class derives from regardless of whether the event channel is in a CORBA or no-CORBA configuration. Aspects then modify the *placeholder class* definition to derive from the appropriate base class. This is demonstrated in the following:

```
public class ProxyPushConsumerImpl
       extends ProxyPushConsumerBase {

       // Appropriate implementation

}

aspect EnableCorba {

       declare parents :
          ProxyPushConsumerBase
             extends ProxyPushConsumerPOA;

       // Other advice

}

aspect DisableCorba {

       declare parents :
          ProxyPushConsumerBase
             implements ProxyPushConsumer;

       // Other advice
}
```

In the above, the `ProxyPushConsumerBase` class is the placeholder class that the aspects modify as necessary, to either extend the `ProxyPushConsumerPOA` class, or implement the `PushConsumer` interface.

With CORBA enabled, it is also necessary to invoke methods on the POA [20] object to obtain a reference from a Servant [20] object. For example, to obtain the `ProxyPushConsumer` interface reference from the servant `ProxyPushConsumerImpl` object, the following is necessary:

```
ProxyPushConsumer ppc =
   ProxyPushConsumerHelper.narrow (
     poa.servant_to_reference (impl));
```

However, in the no-CORBA case, to obtain the interface type reference, the following is sufficient since there are no Servant objects and the implementation class actually implements the `PushConsumer` interface directly:

```
ProxyPushConsumer ppc =
   (ProxyPushConsumer) impl;
```

To transparently provide the correct implementation based on the configuration of the event channel (*i.e CORBA or no-CORBA*) we make use of what we call the "Placeholder Method" pattern. This pattern makes use of a single, empty method for each such operation which is then advised as necessary by aspects. For instance, the placeholder method in this case would be:

```
public ProxyPushConsumer
GetProxyPushConsumerReference (ProxyPushConsumerBase impl)
{
   // Nothing needs to be done here
   return null;
}
```

with the appropriate code weaved in via *around* advice (which is basically an alternate method implementation for the method that we are wrapping around) from the EnableCorba and DisableCorba aspects :

```
aspect EnableCorba {

   ProxyPushConsumer around (...) :
   GetProxyPushConsumerRef (...)
   {
      ProxyPushConsumer ppc =
         ProxyPushConsumerHelper.narrow (
            poa.servant_to_reference (impl));

      return ppc;
   }
}

aspect DisableCorba {

   ProxyPushConsumer around (...) :
   GetProxyPushConsumerRef (...)
   {
      ProxyPushConsumer ppc =
         (ProxyPushConsumer) impl;

      return ppc;
   }
}
```

The advantage of such an approach is that there are only a small number of public interfaces for which this has to be done making it suitable for any application which needs to abstract its use of CORBA this way.

5.4 IDL Generator and Build Process

When different features are enabled in FACET, the IDL of the event channel needs to change accordingly to reflect the presence of those new features, in the case CORBA is enabled (there is no need to generate IDL when CORBA is disabled). In previous work, the changes to the IDL of the event channel were conducted by the use of scripts which makes the changes using primitive text processing and search-and-replace style techniques [15]. However, for our purposes, using such a script would entail having to use different aspects for the cases when CORBA is enabled and when it is not. From a software engineering standpoint, this would be very inefficient and greatly reduce the maintainability of the code.

To address this, we chose to investigate a new technique which involves the generation of the IDL for a given configuration by *reflection* on the classes comprising the event channel's interface. With this, it is only necessary to specify the aspect introductions in the Java code - the corresponding changes to the IDL happen automatically since the mapping from CORBA to Java is well-known.

The generator is run as part of the three-stage build process:

- Aspects that perform introductions are applied to the classes which form the public interface of the event channel
- The IDL Generator reflects on these classes and generates the IDL. The IDL compiler is then run to generate the stub and skeleton classes. In the case CORBA is disabled, this step is automatically skipped.
- All classes comprising the event channel along with the relevant aspects for the particular feature set are compiled and the relevant jUnit [7] tests are run [15].

The IDL Generator we have developed is generic in its implementation and can be used to generate the IDL interfaces corresponding to any set of Java classes. Conceivably, this technique can be extended to any language which has a strong runtime type system and allows reflection.

6. EXPERIMENTAL RESULTS

In this section, we present results that we obtained in estimating the effect that CORBA had on the footprint and throughput performance of FACET. To collect such data, a set of popular configurations was identified based on feedback from several developers of the TAO users community who are using event channels in their application development. In addition, to gauge the effect of individual features on the overall size and performance of the FACET event channel, each feature was studied by measuring its effect across all configurations that included or omitted the given feature.

One method to measure the footprint of a Java application is to sum the size of all the `.class` files that are loaded. Embedded systems that use Java interpreters or just-in-time compilers could use this metric to estimate the amount of RAM needed. Another method consists of generating native code using a compiler (such as GCJ [10]) and then measuring the size of the resulting executable. The compiled code is more suitable for comparisons with C and C++ code. Moreover, embedded real-time applications are likely to precompile to native code for execution predictability. An overall observation has been that the size of the GCJ produced object files are generally larger than the corresponding `.class` files [16]. This is commensurate with the design of `.class` files to be small so as to reduce transmission time over networks.

Here, we report results based on `.class` files that are interpreted and executed using the Sun Java Virtual Machine (JVM) 1.4.0 with Just-In-Time compilation enabled. The experiments were performed on a dual-Xeon processor machine running at 2.40 GHz, with 512 MB of RAM.

With Java and `.class` files, the footprint of the running program increases as classes are loaded. We report the maximum footprint, achieved when all code has been loaded; such measurements are most appropriate for an embedded system. For a native-code compiled version, both the footprint and the resulting throughput are expected to increase.

6.1 Common Configurations

The following are the 10 event channel configurations used in collecting experimental data :

1. *Configuration 1 (Base):* Although the applications requested by developers all required more functionality than the base, it is useful in that it is a lower bound on the footprint. Note that all subsequent tests use the full functionality provided by the base.
2. *Configuration 2:* Several developers only needed configurations similar to the standard CORBA COS Event Service specification. This configuration has CORBA Any payloads and does not support filtering. The pull interfaces were not included in this configuration since they were not used.
3. *Configuration 3:* This configuration is the same as the previous except that the tracing feature is enabled.
4. *Configuration 4:* Structured events and event sets are enabled. This configuration also adds the time to live (TTL) field processing to eliminate loops created by federating event channels. This configuration is still minimal, however, and does not support any kind of event filtering.

5. *Configuration 5:* This configuration has support for dispatching events based on event type. It uses a CORBA octet sequence as the payload type and is a common optimization over using a CORBA Any. This configuration is similar to that used in the TAO Real-Time Event Channel (RTEC).

6. *Configuration 6:* This configuration adds support for the event pull interfaces to configuration 5 and uses a CORBA Any as the payload.

7. *Configuration 7:* This configuration enhances configuration 5 by replacing the simple event type dispatch feature with the event correlation feature. In the corresponding application, event timestamp information was also needed, but the event pull feature was not.

8. *Configuration 8:* This configuration represents one of the largest realistic configurations of FACET. It supports the pull interfaces, uses event correlation, and adds support for statistics collection and reporting. It uses structured events carrying CORBA Any payloads and headers with all possible fields enabled.

9. *Configuration 9:* This configuration adds the tracing feature to configuration 8.

10. *Configuration 10:* This configuration is representative of that used in the Boeing Bold Stroke architecture. It includes a number of features like type filtering, event correlation, event timestamps and the real time dispatcher feature, a feature that allows consumers and suppliers to set real-time priorities on event delivery.

6.2 Footprint Measurements

As shown in Figure 3, the base FACET configuration (config 1) is 3 times larger when CORBA is present: 166,921 bytes with CORBA and 55,250 without.

At the other extreme, one of the fuller-featured FACET configurations (config 9) has a size of 475,100 bytes with CORBA and a size of 342,226 bytes without — approximately 1.4 times larger for CORBA. This is expected since there are a significant number of Stub and Skeleton classes that are generated by the IDL compiler, which are absent in the no-CORBA case.

It must be noted that the size of the ORB has not been included in this study. It follows that if it were indeed included in these measurements, there would be an even bigger difference in the footprint observed. Generally, in full-featured ORBs such as JacORB [3] that are not subsettable, the most casual reference to the ORB causes the entire ORB to be included in the resulting executable code. While ORBs vary in size [9], and some ORBs do offer reduced-feature versions, the choice of which features to include or omit is not made on an application-specific basis. Conceivably, our AOP approach for including features in an event channel could be extended to include only those ORB features needed to support a given event-channel configuration.

Figure 3 shows that the disabling of CORBA for the configurations we considered mostly reduced footprint by about half — appreciable savings for small embedded systems.

6.3 Throughput Measurements

Figure 4 shows the difference in throughput performance with and without CORBA. When configured as the standard CORBA COS Event Service [22], the throughput with CORBA enabled was 1651 events/sec as compared with 131,758 events/sec without — a difference of 2 orders of magnitude! This can be explained by

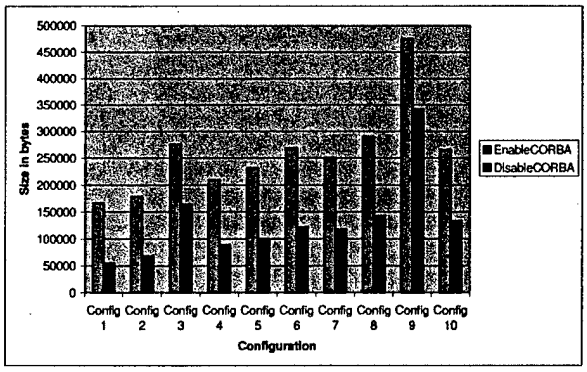

Figure 3: FACET footprint under different configurations

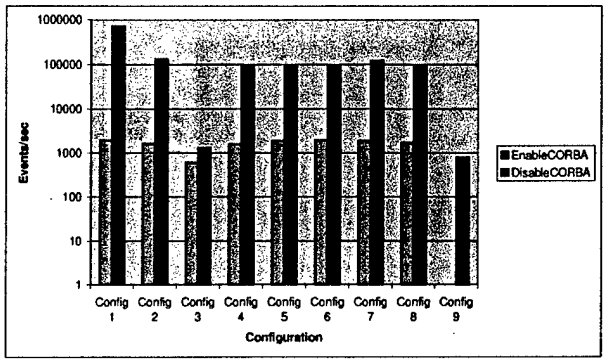

Figure 4: Throughput under different configurations

the fact that the Java ORB, JacORB, does not include optimizations for collocated objects which means that the Stubs and Skeletons perform marshalling and communication over network sockets assuming a truly distributed system. With an ORB such as TAO that does include such optimizations, the performance difference is likely to be less dramatic but still substantial.

This level of improvement without CORBA held for all configurations of the event channel that we studied with the exception of configurations which included the tracing feature (configs 3 and 9). The reason for this can be attributed to the enormous amount of code weaved in by the AspectJ compiler onto all the methods of every class in the event channel, when the tracing feature is enabled. The overhead of these extraneous method calls to the log4j [1] logging library contribute significantly to performance degradation and to the size of the footprint. This observation is consistent with findings in a previous study [16].

6.4 By-Feature Study

We next measured footprint and throughput for various configurations in which only a single feature (and features upon which it depends) was enabled at a time. This experiment quantifies the the size contribution and throughput degradation of a given feature.

Figure 5 shows footprint reduction by-feature, with and without CORBA. For an embedded system, even modest savings can be crucial to a component's cost.

A much greater impact can be seen as we study performance. As shown in Figure 6, the difference for each feature with and without

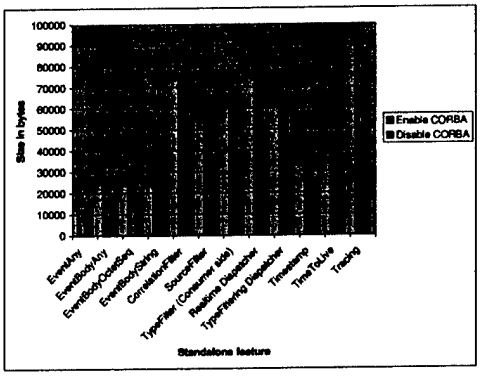

Figure 5: Impact of different features on footprint

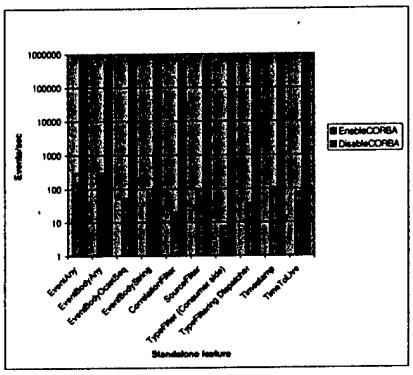

Figure 6: Impact of different features on throughput

CORBA is dramatic. The interesting observation is that no difference is observed *among* the features when CORBA is disabled. An explanation for this is that the aggregation of features on the base and the overhead associated with the code weaved in by the AspectJ compiler is negligible,so that the throughput at this point is limited by the operating system and/or hardware. This indicates that the throughput performance of the event channel with CORBA disabled is at a maximum and is quite unaffected by the feature set (again with the exception of the tracing feature).

It can be argued that a true measure of the average effect of a feature on the footprint and throughput of the event channel can be obtained by measuring the overhead over the set of all possible valid combinations that differ by that one feature [16]. We plan to investigate this line of experimentation in future work. However, when the number of features is large as is the case in FACET's current code base, the number of valid combinations make it quite impractical to run through each one of them. In such cases, a more intelligent method of grouping features is necessary.

7. FUTURE WORK AND CONCLUDING REMARKS

As embedded software continues to grow more complex and participate even more in distributed systems, the need to use standard middleware becomes even more imperative. While frameworks such as ACE and TAO do reasonably given the constraints of embedded systems and meet the needs of existing large-scale embedded systems, small-scale embedded systems need more and the use of AOP does seem promising.

While significant advances have been made in subsetting middleware, with a precise control over footprint and feature set, the use of transport mechanisms such as CORBA is redundant in scenarios where objects are collocated and written in the same language. Building on known AOP techniques, we have abstracted the transport layer of the FACET event channel such that use of CORBA can be specified at build-time thus providing full-customization of an event channel for a particular application. In this paper, we have presented the results of the measurement of the impact of CORBA on the footprint and throughput of the event channel for popular event service configurations presently in use by members of the TAO user community.

As future work, we intend to take the transport layer abstraction further and to support other transport mechanisms such as Java RMI [27] through encapsulation in a feature. We are also investigating extending FACET to make real-time guarantees about event delivery. And finally, we are studying the design patterns involved in building such customizable middleware embedded systems.

8. ACKNOWLEDGEMENTS

We thank Frank Hunleth for answering a number of questions related to his original implementation of FACET, Morgan Deters for providing interesting ideas on the use of aspects and for answering many questions in that connection, Krishnakumar Balasubramanian for general ideas on transport layer abstraction and help with CORBA, and Anand Krishnan for help with various things including proofreading draft versions of this paper.

9. REFERENCES

[1] Apache Software Foundation. log4j. `http://jakarta.apache.org/log4j/`.

[2] Ken Arnold, James Gosling, and David Holmes. *The Java Programming Language*. Addison-Wesley, Boston, 2000.

[3] Gerald Brose. JacORB: Implementation and Design of a Java ORB. In *Proc. DAIS'97, IFIP WG 6.1 International Working Conference on Distributed Aplications and Interoperable Systems*, pages 143–154, September 1997.

[4] Center for Distributed Object Computing. The ACE ORB (TAO). www.cs.wustl.edu/~schmidt/TAO.html, Washington University.

[5] Bryan S. Doerr and David C. Sharp. Freeing Product Line Architectures from Execution Dependencies. In *Proceedings of the 11th Annual Software Technology Conference*, April 1999.

[6] Eric Eide, Kevin Frei, Bryan Ford, Jay Lepreau, and Gary Lindstrom. Flick: A Flexible, Optimizing IDL Compiler. In *Proceedings of ACM SIGPLAN '97 Conference on Programming Language Design and Implementation (PLDI)*, Las Vegas, NV, June 1997. ACM.

[7] Erich Gamma and Kent Beck. JUnit. `www.xProgramming.com/software.htm`, 1999.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[9] Christopher Gill, Venkita Subramonian, Jeff Parsons, Huang-Ming Huang, Stephen Torri, Doug Niehaus, and Douglas Stuart. ORB Middleware Evolution for Networked Embedded Systems. In *Proceedings of the 8th International*

Workshop on Object Oriented Real-time Dependable Systems (WORDS'03), Guadalajara, Mexico, January 2003.

[10] GNU is Not Unix. GCJ: The GNU Complier for Java. `http://gcc.gnu.org/java`, 2002.

[11] Pradeep Gore, Ron K. Cytron, Douglas C. Schmidt, and Carlos O'Ryan. Designing and Optimizing a Scalable CORBA Notification Service. In *Proceedings of the Workshop on Optimization of Middleware and Distributed Systems*, pages 196–204, Snowbird, Utah, June 2001. ACM SIGPLAN.

[12] Timothy H. Harrison, David L. Levine, and Douglas C. Schmidt. The Design and Performance of a Real-time CORBA Event Service. In *Proceedings of OOPSLA '97*, pages 184–199, Atlanta, GA, October 1997. ACM.

[13] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, Reading, MA, 1999.

[14] Frank Hunleth. Building customizable middleware using aspect-oriented programming. Master's thesis, Washington University in Saint Louis, 2002.

[15] Frank Hunleth, Ron Cytron, and Chris Gill. Building Customizable Middleware using Aspect Oriented Programming. In *The OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, Tampa Bay, FL, October 2001. ACM. `http://www.cs.ubc.ca/~kdvolder/Workshops/OOPSLA2001/ASoC.html`.

[16] Frank Hunleth and Ron K. Cytron. Footprint and feature management using aspect-oriented programming techniques. In *Proceedings of the joint conference on Languages, compilers and tools for embedded systems*, pages 38–45. ACM Press, 2002.

[17] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proceedings of the 11th European Conference on Object-Oriented Programming*, June 1997.

[18] Microsoft Corporation. *Distributed Component Object Model Protocol (DCOM)*, 1.0 edition, January 1998.

[19] Object Management Group. *Notification Service Specification*. Object Management Group, OMG Document telecom/99-07-01 edition, July 1999.

[20] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, 2.4 edition, October 2000.

[21] Object Management Group. *The Common Object Request Broker: Architecture and Specification, Revision 2.6*, December 2001.

[22] OMG. *CORBAServices: Common Object Services Specification, Revised Edition*. Object Management Group, 97-12-02 edition, December 1997.

[23] Carlos O'Ryan, Douglas C. Schmidt, and J. Russell Noseworthy. Patterns and Performance of a CORBA Event Service for Large-scale Distributed Interactive Simulations. *International Journal of Computer Systems Science and Engineering*, 17(2), March 2002.

[24] Douglas C. Schmidt. The ADAPTIVE Communication Environment (ACE). www.cs.wustl.edu/~schmidt/ACE.html, 1997.

[25] Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley & Sons, New York, 2000.

[26] David C. Sharp. Reducing Avionics Software Cost Through Component Based Product Line Development. In *Proceedings of the 10th Annual Software Technology Conference*, April 1998.

[27] SUN. Java Remote Method Invocation (RMI) Specification. java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html, 2002.

[28] The AspectJ Organization. Aspect-Oriented Programming for Java. `www.aspectj.org`, 2001.

[29] The Object Management Group. OMG's site for CORBA and UML Success Stories. www.corba.org/, 1999.